A Study of Abramsky's Linear Chemical Abstract Machine

AUTHOR(S):

Mikami, Seikoh; Akama, Yohji

CITATION:

Mikami, Seikoh ...[et al]. A Study of Abramsky's Linear Chemical Abstract Machine. 数理解析研究所講究録 1999, 1096: 68-83

ISSUE DATE:

1999-04

URL:

http://hdl.handle.net/2433/63002

数理解析研究所講究録
1096 巻 1999 年 68-83

A Study of Abramsky's Linear Chemical Abstract Machine*

Seikoh Mikami[1] and Yohji Akama[2] (三上清光, 赤間陽二)

[1] Department of Information Science, Tokyo University,
Hongoh, 7-3-1, Tokyo, Japan, 113-0033
[2] Department of Information Science, Tokyo University,
Hongoh, 7-3-1, Tokyo, Japan, 113-0033 akama@is.s.u-tokyo.ac.jp

Abstract. Abramsky's Linear Chemical Abstract Machine (LCHAM) is a term calculus which corresponds to Linear Logic, via the *Curry-Howard isomorphism*. We introduce a translation from a linear λ-calculus into LCHAM. The translation result can be well regarded as a black box with the i/o ports being atomic. We show that one step computation of LCHAM is equivalent to that of the linear λ-calculus. Then, we prove the *principal typing theorem* of LCHAM, which implies the decidability of type checking.

1 Introduction

There are attempts to regard concurrent computations as chemical reactions. Chemical Abstract Machine (CHAM) [5] is a model of concurrent computation in this line. CHAM influenced on various concurrent calculi such as π-calculus, ambient calculus [6] and join calculus [8].

The points of CHAM is the following:

- once a multiset of objects is applied by a rewriting rule, then the multiset will be consumed and will be transformed to a multiset of objects (in chemistry, a solution of molecules will changes according to chemical reaction laws). In fact, CHAM is resource-sensitive, like Linear Logic (LL).
- a multiset of objects is again an object (in chemistry, a solution encapsulated by a membrane often acts like a molecule). Inside the multiset, computations go through independently. This mechanism may enable us to describe computations inside a sub-network and/or dynamic structuring of networks. The 'membrane' plays an important role in mobile calculi such as ambient calculus and join calculus. CHAM's encapsulation mechanism of computation reminds us of the boxing operation of *proof net* (Girard [9]).

So, we are concerned with Linear Chemical Abstract Machine (Abramsky [1]), which corresponds to LL through *Curry-Howard isomorphism*. Linear Chemical Abstract Machine (LCHAM) consists of not only rewriting rules but also typing rules.

* The preliminary version will appear in the proceedings of TLCA'99.

To investigate computational properties of LCHAM, we introduce a translation from a linear λ-calculus into LCHAM. A linear λ-calculus is a resource-sensitive refinement of λ-calculus. It is employed for analyzing functional programming languages with respect to evaluation strategy [13],[4] and/or resource allocation [7]. We are concerned with a linear λ-calculus which is introduced by Bierman [3], and we translate the terms into proof nets. Then we prove that one step reduction in the linear λ-calculus corresponds to one step reduction in LCHAM modulo a bisimulation.

To investigate type-theoretic properties of LL, we prove the principal typing theorem of LCHAM. The principal typing theorem is an indispensable theorem for implementing a functional language that has a polymorphic type-inference system, such as a programming language ML.

Related Work There are various versions of linear λ-calculi. Abramsky introduced a call-by-value linear λ-calculus [1], Chirimar-Gunter-Riecke introduced a linear λ-calculi with a fix point operator for non-linear function [7].

The linear λ-calculus of this paper was introduced in Bierman et al [3]. Their calculus does not suffer from the coherence problem. Furthermore it has a stable notion of commuting conversions. The commuting-convertible linear λ-terms is translated by our translation into the same proof net. Under the presence of the fix point operator, we don't know how to define the commuting conversion, and how the commuting conversion is related to the structure of proof net.

We introduce a translation from the linear λ-calculus into proof nets, and the translation satisfies the following property: The resulting proof nets can be well regarded as a black box with the i/o ports being 'atomic'. So, such black boxes can be easily connected through their ports. It is not the case in most translation of their multiplicative λ-calculus into proof nets (Bellin-Scott [2], Mackie [11], etc.)

Mackie [12] proved the principal typing theorem of Abramsky's linear λ-calculus. We prove the same theorem for LCHAM in this paper. In proving the principal typing theorems, the reconstruction algorithm of a derivation of a given typing assertion is essential. In the case of linear λ-calculus, the reconstruction algorithm will be deterministic. The type assertions are two-sided sequents $\Gamma \vdash t : A$, and we can only decompose t on their antecedents in reconstructing the derivations.

However, in the case of LCHAM, the reconstruction algorithm will be non-deterministic. Because the type assertions are one-sided sequents like $\vdash t_1 : A_1, \ldots, t_n : A_n$, the reconstruction algorithm choose non-deterministically t_i to decompose. Furthermore, some type-inference rules of LCHAM is another source of non-determinism. So, the existence proof of principal type is not trivial.

Organization In the next section, we review LCHAM [1], a rewriting system for a *proof expression*, which is a representation of a proof of LL. In Section 3, we review the *linear λ-calculus* (Benton et al [3]). We introduce a translation from the linear λ-calculus to LCHAM, and show that one step β-reduction in the linear λ-calculus 'roughly' corresponds to one step reaction rules in LCHAM. In Section

4, we prove the principal typing theorem of LCHAM. To prove this theorem, we introduce *locally correct assertions*, which correspond to proof structures in LL [9], (ordinary type assertions correspond to proof nets of LL).

2 Linear Chemical Abstract Machine

We begin by reviewing Linear Chemical Abstract Machine (LCHAM) [1], a rewriting system representing the cut-elimination procedure of a proof in LL.

*A proof expression (*PEXP*)* is an object to rewrite in LCHAM. *Proof expressions* are defined together with *terms* and *coequations* as follows. Letters $P, Q, \ldots$ stand for PEXPs, $t, u, \ldots$ for terms, $x, y, z, \ldots$ for *names*, and $\bar{x}, \bar{y}, \bar{z}, \ldots$ for lists of names. *Terms* are defined as

$$t ::= x \mid * \mid \odot \mid t_1 \otimes t_2 \mid t_1 \parr t_2 \mid \mathsf{inl}(t) \mid \mathsf{inr}(t) \mid \bar{x}(P \parallel Q) \mid ?t \mid \_ \mid t_1 @ t_2 \mid \bar{x}(P).$$

We call a term of the form $\bar{x}(P)$ or $\bar{x}(P \parallel Q)$ a *closure* and $\bar{x}$ of $\bar{x}(\cdots)$ *binding names*. *Coequations* have the form $t \perp u$, where t and u are terms. Proof expressions have the form $\Theta; \bar{t}$, where Θ is a finite sequence of coequations and is called the *coequation part*. $\bar{t}$ is a finite sequence of terms and is called the *term part*.

2.1 Type inference

Types, ranged over by $A, B, C, \ldots$, are exactly the formulas of LL. For every formula A, its *linear negation* is denoted by $A^{\perp}$. We sometimes write $\bar{t} : \Gamma$ for $t_1 : A_1, \ldots, t_n : A_n$ where $\bar{t} = t_1, \ldots, t_n$ and $\Gamma = A_1, \ldots, A_n$. *Different names are introduced for each instance of the Axiom, With and OfCourse rules.*

$$\frac{\vdash \Theta; \Gamma, u : A, t : B, \Delta}{\vdash \Theta; \Gamma, t : A, u : B, \Delta}\ \text{Exchange} \qquad \frac{}{\vdash ; x : A^{\perp}, x : A}\ \text{Axiom}$$

$$\frac{\vdash \Theta; \Gamma, t : A \quad \vdash \Xi; \Delta, u : A^{\perp}}{\vdash \Theta, \Xi, t \perp u; \Gamma, \Delta}\ \text{Cut}$$

$$\frac{}{\vdash ; * : 1}\ \text{One} \qquad \frac{\vdash \Theta; \Gamma}{\vdash \Theta; \Gamma, \odot : \perp}\ \text{Bot}$$

$$\frac{\vdash \Theta; \Gamma, t : A \quad \vdash \Xi; \Delta, u : B}{\vdash \Theta, \Xi; \Gamma, \Delta, t \otimes u : A \otimes B}\ \text{Times} \qquad \frac{\vdash \Theta; \Gamma, t : A, u : B}{\vdash \Theta; \Gamma, t \parr u : A \parr B}\ \text{Par}$$

$$\frac{\vdash \Theta; \bar{t} : \Gamma, t : A \quad \vdash \Xi; \bar{u} : \Gamma, u : B}{\vdash ; \bar{x} : \Gamma, \bar{x}(\Theta; \bar{t}, t \parallel \Xi; \bar{u}, u) : A \,\&\, B}\ \text{With}$$

$$\frac{\vdash \Theta; \Gamma, t : A}{\vdash \Theta; \Gamma, \mathsf{inl}(t) : A \oplus B}\ \text{Plus-1} \qquad \frac{\vdash \Theta; \Gamma, t : B}{\vdash \Theta; \Gamma, \mathsf{inr}(t) : A \oplus B}\ \text{Plus-2}$$

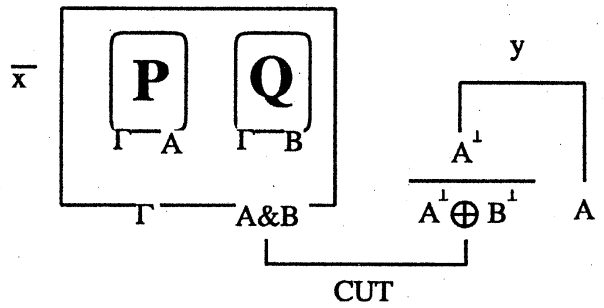
$$\frac{\vdash \Theta; \Gamma}{\vdash \Theta; \Gamma, \_ : ?A}\ \text{Weakening} \qquad \frac{\vdash \Theta; \bar{t} : ?\Gamma, t : A}{\vdash ; \bar{x} : ?\Gamma, \bar{x}(\Theta; \bar{t}, t) : !A}\ \text{OfCourse}$$

$$\frac{\vdash \Theta; \Gamma, t : ?A, u : ?A}{\vdash \Theta; \Gamma, t @ u : ?A}\ \text{Contraction} \qquad \frac{\vdash \Theta; \Gamma, t : A}{\vdash \Theta; \Gamma, ?t : ?A}\ \text{Dereliction}$$

Remark 1. Note that we obtain the rules of LL from the type inference rules by ignoring PEXPs. We can say that $\vdash \Theta;\ \bar{t} : \Gamma$ corresponds to a proof net [9] such that

- the lowest nodes are Γ,
- the Cut-links are represented by Θ, and
- the closures are represented by the boxes.

For example, $\vdash \bar{x}(P \parallel Q) \perp \mathsf{inl}(y);\ \bar{x} : \Gamma, y : A$ represents the following proof net.



2.2 Reductions

Our discussion is limited to linear PEXPs, which we define slightly different from the ones in Abramsky [1]. In our definition, we consider occurrences of names in PEXP only outside closures and not ones in PEXPs inside closures. We consider binding names to be outside the closure.

Definition 1. *A* PEXP $\Theta;\bar{t}$ *is* linear *if and only if*

- *Each name occurring in* $\Theta;\bar{t}$*, does so exactly* twice*;*
- *If a closure* $\bar{x}(\cdots)$ *occurs in* $\Theta;\bar{t}$*, then none of the other occurrences of* $\bar{x}$ *are binding names; and*
- *Each* PEXP *inside a closure is linear.*

We say a PEXP $\Theta;\bar{t}$ is *typable* if and only if $\vdash \Theta;\bar{t} : \Gamma$ is derivable for some Γ. We note that every typable PEXP is linear. Intuitively, the linearity condition of a PEXP means that the PEXP can represent a skeleton of some proof structure [9].

Rewriting rules in LCHAM are classified into *reaction rules* and a *cleanup rule.* The reduction relation determined by the reaction rules is written as $\rightarrow_r$. The reduction relation determined by the cleanup rule is written as $\rightarrow_c$. The reaction rule rewrites only the 'coequations part' of a PEXP.

We regard Θ of a PEXP $\Theta;\bar{t}$ as a *multiset* of coequations, and identify coequations $t \perp u$ and $u \perp t$. We write $\Theta \equiv \Theta'$ if Θ and Θ' are equal in the sense described above. (This corresponds to the "structural rules" in Abramsky [1].) Hereafter, we simply identify Θ and Θ' if $\Theta \equiv \Theta'$. The cleanup rule represents a contraction of a Cut-link involving an Axiom-link.

Cleanup rule.

$$\Theta,\ x \perp t;\ \bar{u} \to_c \Theta;\ \bar{u}[t/x]$$

where x is outside of closures and not a binding variable.

Reaction rule.

Communication	$t \perp x,\ x \perp u \to_r t \perp u$
Unit	$* \perp \odot \to_r$
Pair	$t \otimes u \perp t' \parr u' \to_r t \perp t',\ u \perp u'$
Case Left†	$\bar{x}(\Theta; \bar{t}, t \parallel \Xi; \bar{u}, u) \perp \mathsf{inl}(v) \to_r \Theta,\ \bar{x} \perp \bar{t},\ t \perp v$
Case Right	$\bar{x}(\Theta; \bar{t}, t \parallel \Xi; \bar{u}, u) \perp \mathsf{inr}(v) \to_r \Xi,\ \bar{x} \perp \bar{u},\ u \perp v$
Read	$\bar{x}(\Theta; \bar{t}, t) \perp ?u \to_r \Theta,\ \bar{x} \perp \bar{t},\ t \perp u$
Discard	$\bar{x}(P) \perp \_ \to_r x_1 \perp \_, \ldots, x_n \perp \_$
Copy‡	$\bar{x}(P) \perp u \mathbin{@} v \to_r \bar{x} \perp (\bar{x}^l \mathbin{@} \bar{x}^r),\ \bar{x}(P)^l \perp u,\ \bar{x}(P)^r \perp v$

(†) $\bar{x} \perp \bar{t}$ denotes $x_1 \perp t_1, \ldots, x_n \perp t_n$ if $\bar{x} = x_1, \ldots, x_n$ and $\bar{t} = t_1, \ldots, t_n$.
(‡) $\bar{x}^l$ denotes a list of new names $x_1^l, \ldots, x_n^l$ if $\bar{x} = x_1, \ldots, x_n$, and $\bar{x}(P)^l$ denotes a term where small l's are attached to all names in $\bar{x}(P)$. $\bar{x}^r$ and $\bar{x}(P)^r$ are defined in the same way.

3 Translation From Linear λ-calculus to LCHAM

This section begins with a review of a linear λ-calculus which was introduced by Benton et al[3].

3.1 The Linear λ-calculus

We only consider the $(\multimap, \otimes, !)$-fragment of *intuitionistic linear logic* (ILL).

Types are either a type variable, $A_1 \otimes A_2$, $!A$, or a *linear implication* $A_1 \multimap A_2$.

Pre-linear λ-terms, ranged over by $t, u, \ldots$, are defined as:

$$\begin{aligned} t ::= {} & x \mid t_1 t_2 \mid \lambda x.t \mid t_1 \otimes t_2 \mid \mathsf{let}\ t_1\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ t_2 \\ & \mid \mathsf{promote}\ t_1, \ldots, t_n\ \mathsf{for}\ x_1, \ldots, x_n\ \mathsf{in}\ u \mid \mathsf{derelict}(t) \\ & \mid \mathsf{discard}\ t_1\ \mathsf{in}\ t_2 \mid \mathsf{copy}\ t_1\ \mathsf{as}\ x, y\ \mathsf{in}\ t_2. \end{aligned}$$

Here, *bound* occurrence of variables are either (1) occurrences of x in $(\lambda x.\ \ldots)$, (2) occurrences of x or y in (let t_1 be $x \otimes y$ in $\ldots$) or (copy t_1 as x, y in $\ldots$), or (3) occurrences of $x_1, \ldots, x_n$ in (promote $t_1, \ldots, t_n$ for $x_1, \ldots, x_n$ in $\ldots$). An occurrence of a variable is called *free* if it is not bound. *A linear λ-term* is a pre-linear λ-term t such that each variable occurring free in t does so exactly *once.*

Type inference rules.

$$\frac{}{x : A \vdash x : A}\ Id$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \multimap I \qquad \frac{\Gamma \vdash t : A \multimap B \quad \Delta \vdash u : A}{\Gamma, \Delta \vdash tu : B}\ \multimap E$$

$$\frac{\Gamma \vdash t : A \quad \Delta \vdash u : B}{\Gamma, \Delta \vdash t \otimes u : A \otimes B}\ \otimes I \qquad \frac{\Gamma \vdash t : A \otimes B \quad \Delta, x : A, y : B \vdash u : C}{\Gamma, \Delta \vdash \mathsf{let}\ t\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ u : C}\ \otimes E$$

$$\frac{\Delta_1 \vdash t_1 : !A_1 \quad \cdots \quad \Delta_n \vdash t_n : !A_n \quad x_1 : A_1, \ldots, x_n : A_n \vdash u : B}{\Delta_1, \ldots, \Delta_n \vdash \mathsf{promote}\ t_1, \ldots, t_n\ \mathsf{for}\ x_1, \ldots, x_n\ \mathsf{in}\ u : !B}\ \textit{Promotion}$$

$$\frac{\Gamma \vdash t : !A}{\Gamma \vdash \mathsf{derelict}(t) : A}\ \text{Dereliction} \qquad \frac{\Gamma \vdash t : !A \quad \Delta \vdash u : B}{\Gamma, \Delta \vdash \mathsf{discard}\ t\ \mathsf{in}\ u : B}\ \text{Weakening}$$

$$\frac{\Gamma \vdash t : !A \quad \Delta, x : !A, y : !A \vdash u : B}{\Gamma, \Delta \vdash \mathsf{copy}\ t\ \mathsf{as}\ x, y\ \mathsf{in}\ u : B}\ \text{Contraction}$$

β-reduction of the linear λ-calculus is defined by the following five rewriting rules: $(\lambda x.t)u \to_\beta t[u/x]$,

$$\mathsf{let}\ t \otimes u\ \mathsf{be}\ x \otimes y\ \mathsf{in}\ v \to_\beta v[t/x, u/y],$$
$$\mathsf{derelict}\ (\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u) \to_\beta u[\bar{t}/\bar{x}],$$
$$\mathsf{discard}\ (\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u)\ \mathsf{in}\ v \to_\beta \mathsf{discard}\ \bar{t}\ \mathsf{in}\ v, \text{ and}$$
$$\mathsf{copy}\ (\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u)\ \mathsf{as}\ y^l, y^r\ \mathsf{in}\ s$$
$$\to_\beta \mathsf{copy}\ \bar{t}\ \mathsf{as}\ \bar{z}^l, \bar{z}^r \mathsf{in}\ s\,[\mathsf{promote}\ \bar{z}^l\ \mathsf{for}\ \bar{x}^l\ \mathsf{in}\ u^l\ /\ y^l,\ \mathsf{promote}\ \bar{z}^r\ \mathsf{for}\ \bar{x}^r\ \mathsf{in}\ u^r\ /\ y^r]$$

Here $\bar{t}, \bar{u}, \ldots$ stand for lists of linear λ-terms, and $\bar{x}, \bar{y}, \ldots$ for lists of variables. And if $\bar{x} = x_1, \ldots, x_n$ and $\bar{t} = t_1, \ldots, t_n$, then

$$\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u = \mathsf{promote}\ t_1, \ldots, t_n\ \mathsf{for}\ x_1, \ldots, x_n\ \mathsf{in}\ u$$
$$\mathsf{discard}\ \bar{t}\ \mathsf{in}\ u = \mathsf{discard}\ t_1\ \mathsf{in}\ \cdots \mathsf{discard}\ t_n\ \mathsf{in}\ u$$
$$\mathsf{copy}\ \bar{t}\ \mathsf{as}\ \bar{x}, \bar{y}\ \mathsf{in}\ u = \mathsf{copy}\ t_1\ \mathsf{as}\ x_1, y_1\ \mathsf{in}\ \cdots \mathsf{copy}\ t_n\ \mathsf{as}\ x_n, y_n\ \mathsf{in}\ u$$

3.2 Special Proof Expressions

We translate linear λ-terms into special PEXPs:

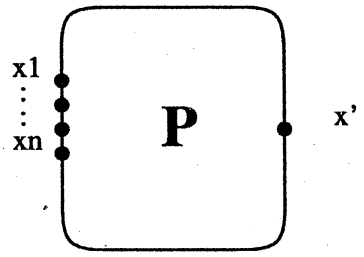
Definition 2 (Special Proof Expressions). *We call a* PEXP $\Theta; \bar{t}$ *a* special proof expression, *or a* special PEXP, *if every term part of $\Theta; \bar{t}$ consists of distinct names.*

The coequation part Θ is sufficient to determine the computational content of the special PEXP $\Theta;\bar{t}$. On the other hand, it is not the case for a usual PEXP; See the following quotation from Abramsky [1]:

> The "molecules" of the linear CHAM are the coequations. We refer to Θ in $\Theta;\bar{t}$ as the "solution", and to $\bar{t}$ as the "main body". The idea is that the computation is done in the solution, with the result recorded in the main body. One can think of each coequation either as a single sequential process, or as a tightly coupled synchronous parallel composition of two processes, proceeding in lockstep. (So coequations could be modelled by "membranes" in Berry and Boudol's terminology; but we shall not pursue this idea.)

We regard the main part $t_1,\ldots,t_n$ of PEXP $\Theta;t_1,\ldots,t_n$ as the ports, and we let the computation results be recorded not on $t_1,\ldots,t_n$ but be recorded in the coequation parts. Moreover, we allow PEXPs to connect each other through their ports. So, we restrict $t_1,\ldots,t_n$ being variables $\bar{x}$. Thus, $\bar{x}$ can be easily interpreted as a list of *port names* in concurrent calculi such as CCS [14]. Therefore, a clear translation of special PEXPs into, for example, agents of CCS may be made easily.

Thus, in the translation of λ-terms, $\Theta;\bar{x},x'$ is interpreted as having $\bar{x}$ as *input ports* and x' as an *output port* as in the following figure.



3.3 The Translation

Basic Idea. Linear λ-terms represent natural deduction style proofs in ILL, while PEXPs represent sequent calculus style proofs in LL (more precisely, an equivalence class of proofs where the equivalence is defined to be "the equality as proof nets"). We adapt Gentzen's translation of natural deduction style proofs into sequent calculus style proofs.

But, we employ a trick to make the translation result a special PEXP. For example, the $(\multimap I)$ rule is translated into $\dfrac{\vdash \Gamma^{\perp}, A^{\perp}, B}{\vdash \Gamma^{\perp}, A^{\perp} \parr B}$ Par (In LL, $A_1 \multimap A_2$ is $A_1^{\perp} \parr A_2$.). If we assign terms to them, then $\dfrac{\bar{x}:\Gamma, x:A \vdash t:B}{\bar{x}:\Gamma \vdash \lambda x.t : A \multimap B}\ \multimap I$ is translated into $\dfrac{\vdash \Theta;\ \bar{x}:\Gamma^{\perp}, x:A^{\perp}, x':B}{\vdash \Theta;\ \bar{x}:\Gamma^{\perp}, x \parr x' : A^{\perp} \parr B}$ Par. However, the lower PEXP $\Theta;\bar{x},x \parr x'$ in the last figure is not a special PEXP, so we let the translation result of $\lambda x.t$ be $\Theta,\ y' \perp x \parr x';\ \bar{x},\ y'.$ with y' being a fresh variable.

This coincides with introducing Cut-rule:

$$\dfrac{\dfrac{\vdash \Theta;\ \bar{x} : \Gamma^{\perp},\ x : A^{\perp},\ x' : B}{\vdash \Theta;\ \bar{x} : \Gamma^{\perp},\ x \parr x' : A^{\perp} \parr B}\ \text{Par} \qquad \vdash ;\ y' : (A^{\perp} \parr B)^{\perp},\ y' : A^{\perp} \parr B}{\vdash \Theta,\ y' \perp x \parr x';\ \bar{x} : \Gamma^{\perp},\ y' : A^{\perp} \parr B}\ \text{Cut}$$

The Translation Rules. For a linear λ-term t, we define its translation result t° by induction on the construction of t. In a PEXP $\Theta; \bar{x}, x'$, we consider Θ to be a multiset of coequations, and $\bar{x}, x'$ as an ordered pair of a *multiset* of names $\bar{x}$ and a name x'.

$$\dfrac{}{x^{\circ} = x \perp x';\ x, x'}\ \text{Id}$$

$$\dfrac{t^{\circ} = \Theta;\ \bar{x}, x, x'}{(\lambda x.t)^{\circ} = \Theta,\ z' \perp x \parr x';\ \bar{x}, z'} \multimap I \qquad \dfrac{t^{\circ} = \Theta;\ \bar{x}, x' \quad u^{\circ} = \Xi;\ \bar{y}, y'}{(tu)^{\circ} = \Theta,\ \Xi,\ x' \perp y' \otimes z';\ \bar{x}, \bar{y}, z'} \multimap E$$

$$\dfrac{t^{\circ} = \Theta;\ \bar{x}, x' \quad u^{\circ} = \Xi;\ \bar{y}, y'}{(t \otimes u)^{\circ} = \Theta,\ \Xi,\ z' \perp x' \otimes y';\ \bar{x}, \bar{y}, z'} \otimes I$$

$$\dfrac{t^{\circ} = \Theta;\ \bar{x}, x' \quad u^{\circ} = \Xi;\ \bar{y}, y_1, y_2, y'}{(\mathsf{let}\ t\ \mathsf{be}\ y_1 \otimes y_2\ \mathsf{in}\ u)^{\circ} = \Theta,\ \Xi,\ x' \perp y_1 \parr y_2;\ \bar{x}, \bar{y}, y'} \otimes E$$

$$\dfrac{t_i^{\circ} = \Theta_i;\ \bar{y}_i, y_i'\ (i = 1, \ldots, n) \qquad u^{\circ} = \Xi;\ x_1, \ldots, x_n, x'}{\begin{array}{l}(\mathsf{promote}\ t_1, \ldots, t_n\ \mathsf{for}\ x_1, \ldots, x_n\ \mathsf{in}\ u)^{\circ} = \\ \quad \Theta_1, \ldots, \Theta_n,\ x_1' \perp y_1', \ldots, x_n' \perp y_n', \\ \quad z' \perp x_1' \cdots x_n'(\Xi, z_1 \perp ? x_1, \ldots, z_n \perp ? x_n; z_1, \ldots, z_n, x'); \\ \quad \bar{y}_1, \ldots, \bar{y}_n, z'\end{array}}\ \text{Promotion}$$

$$\dfrac{t^{\circ} = \Theta;\ \bar{x}, x'}{(\mathsf{derelict}(t))^{\circ} = \Theta,\ x' \perp ?y';\ \bar{x}, y'}\ \text{Dereliction}$$

$$\dfrac{t^{\circ} = \Theta;\ \bar{x}, x' \quad u^{\circ} = \Xi;\ \bar{y}, y'}{(\mathsf{discard}\ t\ \mathsf{in}\ u)^{\circ} = \Theta,\ \Xi,\ x' \perp \_;\ \bar{x}, \bar{y}, y'}\ \text{Discard}$$

$$\dfrac{t^{\circ} = \Theta;\ \bar{x}, x' \quad u^{\circ} = \Xi;\ \bar{y}, y_1, y_2, y'}{(\mathsf{copy}\ t\ \mathsf{as}\ y_1, y_2\ \mathsf{in}\ u)^{\circ} = \Theta,\ \Xi,\ x' \perp y_1 \mathbin{@} y_2;\ \bar{x}, \bar{y}, y'}\ \text{Copy}$$

3.4 The Computational Properties

The set of all the special PEXPs is not closed under the cleanup rule, Fortunately, the cleanup rule is not so important when considering its computational meaning. Instead of the cleanup rule, we define several concepts about special PEXPs. *In the rest of this section, we consider only* linear *special* PEXPs.

Definition 3. *On the set of all the linear special* PEXPs, *we define* $\cong$ *to be the smallest equivalence relation satisfying:*

(1) $P[z/x] \cong P$, *for a fresh name* z.
(2) $\Theta, y \perp z; \bar{x}, x' \cong \Theta[y/z]; \bar{x}, x'$.
(3) $\Theta, x \perp \_; \bar{x}, x, x' \cong \Theta; \bar{x}, x'$.
(4) $\Theta, x \perp \odot; \bar{x}, x, x' \cong \Theta; \bar{x}, x'$.

Clause (2) is sufficient to handle a cleanup rule;

$\Theta, y' \perp x'; \bar{x}, y' \overset{by(2)}{\cong} \Theta[y'/x']; \bar{x}, y' \overset{by(1)}{\cong} \Theta; \bar{x}, x'$. Clauses (3) and (4) are required because free variables often disappear via β-reduction in λ-calculus. (In fact, clause (4) is not needed here, but if we accept clause (3), it is unnatural not to accept clause (4).)

Definition 4. *Define* $P \Rightarrow_r Q \overset{def}{\Longleftrightarrow} P \rightarrow_{r0}^* \rightarrow_{r1} \rightarrow_{r0}^* Q$. *Here* $\rightarrow_{r0}^*$ *is a reflexive and transitive closure of* $\rightarrow_{r0}$. *The* $\rightarrow_{r0}$ *is determined by the communication rule, and* $\rightarrow_{r1}$ *by the other reaction rules.*

Proposition 1. *The translation result of any linear* λ*-term is normal with respect to* $\rightarrow_{r1}$.

Proposition 2. $\cong$ *is a* bisimulation *with respect to* $\Rightarrow_r$, *that is, if* $P \cong Q$ *and* $P \Rightarrow_r P'$, *then some* Q' *satisfies* $P' \cong Q'$ *and* $Q \Rightarrow_r Q'$.

Proof. In view of the linearity of P and Q, it is easily shown that $\cong$ is a bisimulation with respect to $\rightarrow_{r0}^*$ and $\rightarrow_{r1}$, from which the proposition follows directly.

Corollary 1. *If* $P \cong \Rightarrow_r Q$, *then* $P \Rightarrow_r \cong Q$.

The relation $\cong$ is 'compatible' with the translation. For example, if $\Theta; \bar{x}, x, x' \cong \Xi; \bar{y}, x, y'$, then $\Theta, z' \perp x \parr x'; \bar{y}, z' \cong \Xi, z' \perp x \parr y'; \bar{y}, z'$ holds. In particular, if $t^\circ \cong u^\circ$, then $(\lambda x.t)^\circ \cong (\lambda x.u)^\circ$ and so forth.

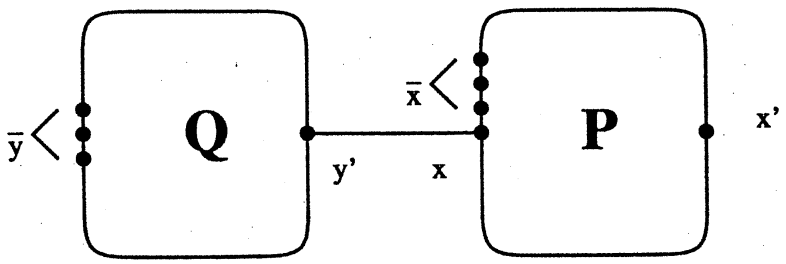
Next, we prove the following theorem:

Theorem 1. *Let* t *and* u *be linear* λ*-terms. If* $t \rightarrow_\beta u$, *then* $t^\circ \Rightarrow_r \cong u^\circ$, *i.e,* t° *goes to a term which is* $\Rightarrow_r \cong$ *to* u°.

To verify the theorem, we define a concept which corresponds to substitution.

Definition 5. *For* $P = \Theta;\ \bar{x}, x, x'$ *and* $Q = \Xi;\ \bar{y}, y'$, *we define* $P[x \leftarrow Q] \overset{def}{=} \Theta, \Xi, x \perp y';\ \bar{x}, \bar{y}, x'$.

Intuitively, $P[x \leftarrow Q]$ is a process where an "input port" x of P is connected to the "output port" of Q. The following figure illustrates this.



Proposition 3. *For all linear* λ*-terms* t *and* u, *and for all free variable* x *in* t, $(t[u/x])^\circ \cong t^\circ[x \leftarrow u^\circ]$.

Proof. Let $u^\circ = \Xi; \bar{y}, y'$. The proof is done by induction on the construction of t. Note that x occurs in t exactly once. In this proof, the 'compatibility' of $\cong$ with the translation described above is used.

Proposition 4. *For each rewriting rule $l \to_\beta r$ of β-reduction, we have $l^\circ \Rightarrow_r \cong r^\circ$. That is,*

$$\begin{aligned}
((\lambda x.t)u)^\circ &\Rightarrow_r \cong (t[u/x])^\circ.\\
(\mathsf{let}\ t_1 \otimes t_2\ \mathsf{be}\ y_1 \otimes y_2\ \mathsf{in}\ u)^\circ &\Rightarrow_r \cong (u[t_1/y_1, t_2/y_2])^\circ.\\
(\mathsf{derelict}\ (\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u))^\circ &\Rightarrow_r \cong (u[\bar{t}/\bar{x}])^\circ.\\
(\mathsf{discard}\ (\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u)\ \mathsf{in}\ s)^\circ &\Rightarrow_r \cong (\mathsf{discard}\ \bar{t}\ \mathsf{in}\ s)^\circ.\\
(\mathsf{copy}\ (\mathsf{promote}\ \bar{t}\ \mathsf{for}\ \bar{x}\ \mathsf{in}\ u)\ \mathsf{as}\ y^l, y^r\ \mathsf{in}\ s)^\circ &\Rightarrow_r \cong (\mathsf{copy}\ \bar{t}\ \mathsf{as}\ \bar{z}^l, \bar{z}^r\ \mathsf{in}\\
&\qquad s[\mathsf{promote}\ \bar{z}^l\ \mathsf{for}\ \bar{x}^l\ \mathsf{in}\ u^l\ /\ y^l,\\
&\qquad\ \mathsf{promote}\ \bar{z}^r\ \mathsf{for}\ \bar{x}^r\ \mathsf{in}\ u^r\ /\ y^r])^\circ.
\end{aligned}$$

Proof. For the proof of the first claim, let $t^\circ = \Theta;\ \bar{x}, x, x'$ and $u^\circ = \Xi;\ \bar{y}, y'$. Then, $(\lambda x.t)^\circ = \Theta;\ z' \perp x \parr x';\ \bar{x}, z'$ and so

$$\begin{aligned}
((\lambda x.t)u)^\circ &= \Theta, \Xi, z' \perp x \parr x', z' \perp y' \otimes w';\ \bar{x}, \bar{y}, w'\\
&\to_{r0} \Theta, \Xi, x \parr x' \perp y' \otimes w';\ \bar{x}, \bar{y}, w'\\
&\to_{r1} \Theta, \Xi, x \perp y', x' \perp w';\ \bar{x}, \bar{y}, w'\\
&\cong \Theta, \Xi, x \perp y';\ \bar{x}, \bar{y}, x' = t^\circ[x \leftarrow u^\circ] \cong (t[u/x])^\circ
\end{aligned}$$

The last is by Proposition 3. The other four claims can be proved similarly.

The proof of Theorem 1 is by induction on the derivation of $t \to_\beta u$.

From Theorem 1, we can conclude that the β-reductions in linear λ-calculus roughly correspond to the reaction rules except the communication rule in LCHAM.

The commuting conversion $\to_c$ is defined as follows. Let $f(t)$ stand for either (let s be $x \otimes y$ in t), (discard s in t), or (copy s as x, y in t). And let $g(t)$ stand for either (tu), (let t be $z_1 \otimes z_2$ in u), (discard t in u), (copy t as z_1, z_2 in u), or (derelict(t)). Then, the commuting conversion is by definition $g(f(t)) \to_c f(g(t))$. For example, (let s be $x \otimes y$ in $t)u \to_c$ let s be $x \otimes y$ in tu. The commuting conversions expose 'hidden' redexes in terms.

We can prove that commuting-convertible linear λ-terms are identified when translated into PEXP. More precisely,

Proposition 5. *If t and u are linear λ-terms and $t \to_c u$, then $t^\circ \cong u^\circ$.*

Proof. We have only to check all the entries of commuting conversions. For example, if $s^\circ = \Theta; \bar{x}, x'$, $t^\circ = \Xi; \bar{y}, y'$ and $u^\circ = \Pi; \bar{z}, z'$, then both ((discard s in $t)u)^\circ$ and (discard s in $tu)^\circ$ turn out to be $\Theta, \Xi, \Pi, x' \perp \_, y' \perp z' \otimes w';\ \bar{x}, \bar{y}, \bar{z}, w'$. Thus, the translation is preserved via a commuting conversion (discard s in $t)u \to_c$ discard s in tu. The other cases are all done in the same way.

Proposition 6 (M. Hasegawa [10]). *Let MLL stand for multiplicative linear logic and let MILL stand for multiplicative intuitionistic linear logic. Then, for any term $\vdash M : \sigma^\circ$ of MLL with σ being a MILL-definable type, there exists a term $\vdash N : \sigma$ of MILL such that $\vdash M = N^\circ$ in MLL.*

So,

Proposition 7. *If t and u are linear λ-terms and $t^\circ \cong u^\circ$, then $t =_c u$.*

4 Principal Typing Theorem of LCHAM

Next, we prove the *principal typing theorem* of LCHAM:

Theorem 2 (Principal Typing). *There is an algorithm such that given a* PEXP *P,*

1. *if P is typable, then it computes a principal type,*
2. *or else it terminates by outputting "not typable".*

Here,

Definition 6 (Principal Typing). *We write $\Vdash \Theta; \bar{t} : \Gamma$, when for all Δ, these are equivalent: (1) $\vdash \Theta; \bar{t} : \Delta$, and (2) $\Delta = \Gamma\sigma$ for some substitution σ.*

Γ is called a principal type *of $\Theta; \bar{t}$. It is easy to see that Γ is unique up to renaming of type variables. Hereafter, we write $pt(\Theta; \bar{t})$ to represent Γ.*

In LCHAM, a type-assertion may have many derivations, unlike a type system of λ-calculus. In particular, a type-assertion $\vdash \Theta, \Xi; \Gamma, \Delta, t \otimes u : A \otimes B$ can be inferred from $\vdash \Theta; \Gamma, t : A$ and $\vdash \Xi; \Delta, u : B$ by an inference rule R=Times, but it may also be inferred by R from another $\vdash \cdots, t : A$ and $\vdash \cdots, u : B$. The same annoyance arises when R is a Cut-rule.This is why the algorithm we will construct in the proof is non-deterministic, while the algorithm for principal types of λ-terms is deterministic.

In Subsection 4.3, we will present the algorithm, and will prove the termination property and the correctness. The correctness proof consists of the verification of Theorem 2 (1) and (2). Theorem 2 (1) will be proved by using the Principal Inference Lemma (i.e., Proposition 9 and 13) in Subsection 4.1, and (2) will be proved by using the Generation Lemma (i.e., Proposition 12 and Proposition 8) in Subsection 4.2.

Hereafter, for sequences Γ and Δ of formulas, we denote by $mgu(\Gamma; \Delta)$ a most general unifier θ such that $\Gamma\theta = \Delta\theta$. Note that it is computable.

4.1 Easy Part of the Proof

Proposition 8 (Generation Lemma, part 1).

1. *If $\vdash \Theta; \Gamma, t_1 \parr t_2 : C$, then C is of the form $A \parr B$ and $\vdash \Theta; \Gamma, t_1 : A, t_2 : B$.*
2. *If $\vdash \Theta; \Gamma, \mathsf{inl}(t) : C$, then C is of the form $A \oplus B$ and $\vdash \Theta; \Gamma, t : A$.*

3. *If* $\vdash \Theta;\ \Gamma, \mathsf{inr}(t) : C$, *then* C *is of the form* $A \oplus B$ *and* $\vdash \Theta;\ \Gamma, t : B$.
4. *If* $\vdash \Theta;\ \Gamma, \odot : C$, *then* $C = \bot$ *and* $\vdash \Theta;\ \Gamma$.
5. *If* $\vdash \Theta;\ \Gamma, \_ : C$, *then* C *is of the form* $?A$ *and* $\vdash \Theta;\ \Gamma$.
6. *If* $\vdash \Theta;\ \Gamma, ?t : C$, *then* C *is of the form* $?A$ *and* $\vdash \Theta;\ \Gamma, t : A$.
7. *If* $\vdash \Theta;\ \Gamma, t_1 \mathbin{@} t_2 : C$, *then* C *is of the form* $?A$ *and* $\vdash \Theta;\ \Gamma, t_1 : C, t_2 : C$.
8. *If* $\vdash ;\ \bar{x} : \Gamma, \bar{x}(\Theta; \bar{t}, t) : A$, *then for some* Γ', A', *we have* $\Gamma = ?\Gamma'$ *and* $A = !A'$ *and* $\vdash \Theta;\ \bar{t} : \Gamma, t : A'$.
9. *If* $\vdash ;\ \bar{x} : \Gamma, \bar{x}(\Theta_1; \bar{t_1}, u_1 \parallel \Theta_2; \bar{t_2}, u_2) : C$, *then* C *is of the form* $A_1 \,\&\, A_2$, *and* $\vdash \Theta_i;\ \bar{t_i} : \Gamma, u_i : A_i$ *for* $i = 1, 2$.

Proposition 9 (Principal Type Inference, part 1). *The following are admissible inference rules.*

$$\frac{\Vdash \Theta;\ \Gamma, t : A, u : B}{\Vdash \Theta;\ \Gamma, t \parr u : A \parr B} \qquad \frac{\Vdash \Theta;\ \Gamma, t : A}{\Vdash \Theta;\ \Gamma, \mathsf{inl}(t) : A \oplus \alpha}\ \dagger \qquad \frac{\Vdash \Theta;\ \Gamma, t : A}{\Vdash \Theta;\ \Gamma, \mathsf{inr}(t) : \alpha \oplus A}\ \dagger.$$

$$\frac{\Vdash \Theta;\ \Gamma}{\Vdash \Theta;\ \Gamma, \odot : \bot} \qquad \frac{\Vdash \Theta;\ \Gamma}{\Vdash \Theta;\ \Gamma, \_ : ?\alpha}\ \dagger \qquad \frac{\Vdash \Theta;\ \Gamma, t : A}{\Vdash \Theta;\ \Gamma,\ ?t : ?A.}$$

$$\frac{\Vdash \Theta;\ \Gamma, t : A, u : B \qquad \mu = mgu(A; B),\ \nu = mgu(?\alpha; A\mu)\ \mathit{exist}}{\Vdash \Theta;\ \Gamma\mu\nu,\ t \mathbin{@} u : A\mu\nu}\ \dagger.$$

$$\frac{\Vdash \Theta;\ \bar{t} : \Gamma, t : A \quad \mu = mgu(\Gamma; ?\bar{\alpha})\ \mathit{exists}}{\Vdash ;\ \bar{x} : \Gamma\mu,\ \bar{x}(\Theta; \bar{t}, t) : !A\mu}\ \dagger.$$

$$\frac{\Vdash \Theta;\ \bar{t} : \Gamma, t : A \quad \Vdash \Xi;\ \bar{u} : \Gamma', u : B \quad \mu = mgu(\Gamma; \Gamma')\ \mathit{exists}}{\Vdash ;\ \bar{x} : \Gamma\mu,\ \bar{x}(\Theta; \bar{t}, t \parallel \Xi; \bar{u}, u) : (A \,\&\, B)\mu}\ \P$$

(†) α *is a fresh type variable. (¶) The premises of the form* $\Vdash \cdots$ *share no variables.*

4.2 Difficult Part of the Proof

As we explained in Section 2, a linear PEXP represents a skeleton of a proof structure. It is well-known that correctness of a proof structure depends mainly on the *skeleton*. We introduce *locally correct assertions*, which correspond to 'proof structures.'

Definition 7 (Locally Correct Assertion). *An assertion* $\vdash_l \Theta;\ \bar{t} : \Gamma$, *which we call a* locally correct assertion, *holds if and only if it is derivable in the inference system* LCHAM′. *Here* LCHAM′ *is obtained from* LCHAM *by replacing the Cut-rule and the Times-rule with the following four rules:*

$$\frac{}{\vdash_l ;} \qquad \frac{\vdash_l \Theta;\ \Gamma \quad \vdash_l \Xi;\ \Delta}{\vdash_l \Theta, \Xi;\ \Gamma, \Delta}\ \mathit{Mix} \qquad \frac{\vdash_l \Theta;\ \Gamma, t : A, u : A^{\perp}}{\vdash_l \Theta, t \perp u;\ \Gamma} \qquad \frac{\vdash_l \Theta;\ \Gamma, t : A, u : B}{\vdash_l \Theta;\ \Gamma, t \otimes u : A \otimes B}$$

Intuitively, the derivation of $\vdash_l \Theta;\ \bar{t} : \Gamma$ corresponds to a *proof structure* [9] with conclusions Γ. It is easy to see the following:

Proposition 10. *1. If* $\vdash_l \Theta;\ \Gamma, u_1 \otimes u_2 : C$, *then* C *must be of the form* $A \otimes B$ *and* $\vdash_l \Theta;\ \Gamma, u_1 : A, u_2 : B$.
2. If $\vdash_l \Theta, u_1 \perp u_2;\ \Gamma$, *then there is an* A *such that* $\vdash_l \Theta;\ \Gamma, u_1 : A^{\perp}, u_2 : A$.

Proposition 11. *If* $\vdash_l \Theta;\ \Gamma$ *and* $\Theta \supseteq \Theta'$, $\Gamma \supseteq \Gamma'$, *then* $\vdash_l \Theta';\ \Gamma'$; *provided that* $\Theta'; \Gamma' = \Theta'; \vec{t} : \Gamma''$ *for some linear* PEXP $\Theta'; \vec{t}$.

Proof. By induction on the deduction of $\vdash_l \Theta;\ \Gamma$.

Theorem 3. *For a typable* PEXP $\Theta;\ \bar{t}$, $\vdash \Theta;\ \bar{t} : \Gamma$ *if and only if* $\vdash_l \Theta;\ \bar{t} : \Gamma$.

Proof. The only-if part. Note that for each rule $\dfrac{\vdash Q_1 \cdots \vdash Q_n}{\vdash P}$ of the system $\vdash$, if in the system $\vdash_l$ we assume $\vdash_l Q_1 \cdots \vdash_l Q_n$ as axioms, we can infer $\vdash_l P$ by using the Mix-rule. Therefore, we are done. *The if part.* Because $\Theta;\ \bar{t}$ is typable, there is some Δ such that $\vdash \Theta;\ \bar{t} : \Delta$. The proof is by induction on the height of this derivation.

If the last rule is the Times-rule, the derivation ends with

$$\frac{\vdash \Theta_i;\ \bar{t}_i : \Delta_i, u_i : A_i\ (i = 1, 2)}{\vdash \Theta_1, \Theta_2;\ \bar{t}_1 : \Delta_1, \bar{t}_2 : \Delta_2, u_1 \otimes u_2 : A_1 \otimes A_2}\ .$$

By Proposition 10, $\vdash_l \Theta;\ \bar{t} : \Gamma$ must be of the form $\vdash_l \Theta;\ \bar{t}_1 : \Gamma_1, \bar{t}_2 : \Gamma_2, u_1 \otimes u_2 : A_1' \otimes A_2'$. Moreover, $\vdash_l \Theta;\ \bar{t}_1 : \Gamma_1, \bar{t}_2 : \Gamma_2, u_1 : A_1', u_2 : A_2'$. We note that each $\Theta_i; \bar{t}_i, u_i$ is linear. Hence, by Proposition 11, $\vdash_l \Theta_i;\ \bar{t}_i : \Gamma_i, u_i : A_i'$. By induction hypotheses, $\vdash \Theta_1;\ \bar{t}_1 : \Gamma_1, u_1 : A_1'$ and $\vdash \Theta_2;\ \bar{t}_2 : \Gamma_2, u_2 : A_2'$. Then, by applying the Times-rule we can conclude $\vdash \Theta_1, \Theta_2;\ \bar{t}_1 : \Gamma_1, \bar{t}_2 : \Gamma_2, u_1 \otimes u_2 : A_1' \otimes A_2'$, i.e. $\vdash \Theta;\ \bar{t} : \Gamma$. The other cases are easy and similar.

Proposition 12 (Generation Lemma, part 2).

1. *Suppose some deduction ends with* $\dfrac{\vdash \Theta_1;\ \bar{t}_1 : \Gamma_1, u_1 : A_1 \quad \vdash \Theta_2;\ \bar{t}_2 : \Gamma_2, u_2 : A_2}{\vdash \Theta;\ \bar{t} : \Gamma, u_1 \otimes u_2 : A_1 \otimes A_2}$. *Then if* $\vdash \Theta_1, \Theta_2;\ \bar{t}_1 : \Delta_1, \bar{t}_2 : \Delta_2, u_1 \otimes u_2 : C$, *then* C *is of the form* $B_1 \otimes B_2$ *and* $\vdash \Theta_i;\ \bar{t}_i : \Delta_i, u_i : B_i$ $(i = 1, 2)$.
2. *Suppose some deduction ends with* $\dfrac{\vdash \Theta_1;\ \bar{t}_1 : \Gamma_1, u_1 : A \quad \vdash \Theta_2;\ \bar{t}_2 : \Gamma_2, u_2 : A^{\perp}}{\vdash \Theta, u_1 \perp u_2;\ \bar{t} : \Gamma}$. *If* $\vdash \Theta_1, \Theta_2, u_1 \perp u_2;\ \bar{t}_1 : \Delta_1, \bar{t}_2 : \Delta_2$, *there is a* B *such that* $\vdash \Theta_1;\ \bar{t}_1 : \Delta_1, u_1 : B$ *and* $\vdash \Theta_2;\ \bar{t}_2 : \Delta_2, u_2 : B^{\perp}$.

Proof. The premise implies through Theorem 3 that $\vdash_l \Theta_1, \Theta_2; \bar{t}_1 : \Gamma_1, \bar{t}_2 : \Gamma_2, u_1 \otimes u_2 : C$. By Proposition 10, $\vdash_l \Theta_1, \Theta_2; \bar{t}_1 : \Gamma_1, \bar{t}_2 : \Gamma_2, u_1 : B_1, u_2 : B_2$ and $C = B_1 \otimes B_2$ for some B_1 and B_2. Because the premise (i) : $\vdash \Theta_i; \bar{t}_i : \Gamma_i, u_i : A_i$ implies the linearity of $\Theta_i; \bar{t}_i, u_i$, Proposition 11 implies $\vdash_l \Theta_i;\ \bar{t}_i : \Delta_i, u_i : B_i$, and because of (i), Theorem 3 implies $\vdash \Theta_i;\ \bar{t}_i : \Delta_i, u_i : B_i$. The second claim can be proved in the same way as above.

Proposition 13 (Principal Type Inference, part 2). *The following are admissible inference rules.*

$$\frac{\Vdash \Theta_i;\ \bar{t_i} : \Gamma_i,\ u_i : A_i\ (i = 1,2)}{\Vdash \Theta_1, \Theta_2;\ \bar{t_1} : \Gamma_1,\ \bar{t_2} : \Gamma_2,\ u_1 \otimes u_2 : A_1 \otimes A_2}$$

$$\frac{\Vdash \Theta_i;\ \bar{t_i} : \Gamma_i,\ u_i : A_i\ (i = 1,2) \quad \mu = mgu(A_1; A_2^{\perp})\ \textit{exists}}{\Vdash \Theta_1, \Theta_2, u_1 \perp u_2;\ \bar{t_1} : \Gamma_1\mu,\ \bar{t_2} : \Gamma_2\mu}$$

The premises of the form $\Vdash \cdots$ *share no variables.*

Proof. To prove the admissibility of the first inference rule, let $\vdash \Theta_1, \Theta_2;\ \bar{t_1} : \Delta_1, \bar{t_2} : \Delta_2, u_1 \otimes u_2 : C$. The premise of the rule implies the existence of a deduction ending with $\dfrac{\vdash \Theta_i;\ \bar{t_i} : \Gamma_i,\ u_i : A_i\ (i = 1,2)}{\vdash \Theta_1, \Theta_2;\ \bar{t_1} : \Gamma_1, \bar{t_2} : \Gamma_2, u_1 \otimes u_2 : A_1 \otimes A_2}$. By Proposition 12, C is of the form $A_1' \otimes A_2'$ and $\vdash \Delta_i;\ \bar{t_i} : \Gamma_i, u_i : A_i'$ with $i = 1,2$. Thus, for some σ_i, $\Delta_i = \Gamma_i\sigma_i$ and $A_i' = A_i\sigma_i$. Because of the side condition, we have $\Delta_i = \Gamma_i\sigma$ and $C\sigma = (A_1 \otimes A_2)\sigma$ for σ being defined below. If α occurs in Γ_i, A_i, then $\sigma(\alpha)$ is $\sigma_i(\alpha)$, or else it is α. Hence we are done. The admissibility of the second inference rule can be shown in the same way.

4.3 The Algorithm for Principal Type

To compute $pt(P)$, *do the following:*

1. *If* P *is of the form* $\Theta; \bar{t}, t_1 \parr t_2$, *then: if* $pt(\Theta; \bar{t}, t_1, t_2) = [\Gamma, A, B]$, *then* $pt(P) = [\Gamma, A \parr B]$, *else failure.*
2. *If* P *is of the form* $\Theta; \bar{t}, \mathsf{inl}(t)$, *then: if* $pt(\Theta; \bar{t}, t) = [\Gamma, A]$, *then* $pt(P) = [\Gamma, A \oplus \alpha]$, *else failure where* α *is a fresh type variable.*
3. *If* P *is of the form* $\Theta; \bar{t}, \mathsf{inr}(t)$, *then: if* $pt(\Theta; \bar{t}, t) = [\Gamma, A]$, *then* $pt(P) = [\Gamma, \alpha \oplus A]$, *else failure where* α *is a fresh type variable.*
4. *If* P *is of the form* $\Theta; \bar{t}, \odot$, *then: if* $pt(\Theta; \bar{t}) = [\Gamma]$, *then* $pt(P) = [\Gamma, \perp]$, *else failure.*
5. *If* P *is of the form* $\Theta; \bar{t}, \_$, *then: if* $pt(\Theta; \bar{t}) = [\Gamma]$, *then* $pt(P) = [\Gamma, ?\alpha]$, *else failure. Here* α *is a fresh type variable.*
6. *If* P *is of the form* $\Theta; \bar{t}, ?t$, *then: if* $pt(\Theta; \bar{t}, t) = [\Gamma, A]$, *then* $pt(P) = [\Gamma, ?A]$, *else failure.*
7. *If* P *is of the form* $\Theta; \bar{t}, t_1 \mathbin{@} t_2$, *then: if* $pt(\Theta; \bar{t}, t, u) = [\Gamma, A, B]$ *and both of* $\mu = mgu(A; B)$ *and* $\nu = mgu(?\alpha; A\mu)$ *exist, then* $pt(P) = \Gamma\mu\nu, A\mu\nu$, *else failure.*
8. *If* P *is of the form* $;*$, *then* $pt(P) = [1]$.
9. *If* P *is of the form* $;x,x$, *then* $pt(P) = [\alpha^{\perp}, \alpha]$, *where* α *is a fresh type variable.*
10. *If* P *is of the form* $;\bar{x}, \bar{x}(Q)$, *then: if* $pt(Q) = [\Gamma, A]$, *and if* $\mu = mgu(\Gamma; ?\bar{\alpha})$ *exists (where* $\bar{\alpha}$ *is a list of fresh names), then* $pt(P) = [\Gamma\mu, !A\mu]$. *Otherwise, failure.*

11. *If P is of the form $;\bar{x},\bar{x}(Q \parallel Q')$, then: if $pt(Q) = [\Gamma_1, A]$, $pt(Q') = [\Gamma_2, B]$, and $\mu = mgu(\Gamma_1; \Gamma_2)$ exists, then $pt(P) = [\Gamma_1\mu, (A \,\&\, B)\mu]$. Otherwise, failure.*
12. *Otherwise, let $P = \Theta;\bar{t}$. For every decomposition of the form $\Theta = \Theta_1, \Theta_2$ and $\bar{t} = \bar{t_1}, \bar{t_2}, u_1 \otimes u_2$, try to compute $pt(\Theta_1; \bar{t_1}, u_1)$ and $pt(\Theta_2; \bar{t_2}, u_2)$. If it fails for every decomposition, go to 13. If it succeeds for a decomposition, let the result be $[\Gamma, A]$ and $[\Delta, B]$. Then, $pt(P) = [\Gamma, \Delta, A \otimes B]$.*
13. *For every decomposition of the form $\Theta = \Theta_1, \Theta_2, u_1 \perp u_2$ and $\bar{t} = \bar{t_1}, \bar{t_2}$, try to compute $pt(\Theta_1; \bar{t_1}, u_1)$ and $pt(\Theta_2; \bar{t_2}, u_2)$. If it succeeds for a decomposition, let the result be $[\Gamma, A]$ and $[\Delta, B]$. If $mgu(A; B^{\perp})$ exists, then $pt(P) = [\Gamma\mu, \Delta\mu]$. Otherwise, failure.*

This algorithm terminates for any input, because the number of constructors in P decreases strictly at each step. Moreover, the correctness of each step is verified as follows: When P is typable, let π be a derivation of it. Then we can show that $pt(P)$ is a principal type of P, by induction on π, by using Proposition 9 and Proposition 13. When P is not typable, it outputs "failure," because of Proposition 12 and Proposition 8. Thus, the proof of Theorem 2 is completed.

5 Principal Typing Theorem of the Linear λ-calculus

We are interested in whether our translation preserves(and/or reflects) the principal type. So, we first prove that the linear λ-calculus of this paper admits the principal typing theorem.

As before, we prove that the generation lemma and the admissibility of principal type inferences. The generation lemma is easy. We consider the following principal type inferences:

$$\frac{\begin{array}{c}\Delta_i \Vdash t_i : A_i \ (i = 1, \ldots, n) \\ x_1 : C_1, \ldots, x_n : C_n \Vdash u : B \\ \theta = mgu(A_1, \ldots, A_n;\ !C_1, \ldots, !C_n)\end{array}}{\Delta_1\theta, \ldots, \Delta_n\theta \Vdash \mathsf{promote}\ t_1, \ldots, t_n\ \mathsf{for}\ x_1, \ldots, x_n\ \mathsf{in}\ u : !B\theta}\ ¶$$

$$\frac{\Gamma \Vdash t : A \quad \Delta \Vdash u : B}{\Gamma, \Delta \Vdash \mathsf{discard}\ t\ \mathsf{in}\ u : B}\ ¶$$

$$\frac{\Gamma \Vdash t : A_0 \quad \Delta, x : A_1, y : A_2 \Vdash u : B \quad \theta = mgu(A_0, A_0;\ A_1, A_2)}{\Gamma\theta, \Delta\theta \Vdash \mathsf{copy}\ t\ \mathsf{as}\ x, y\ \mathsf{in}\ u : B\theta}\ ¶$$

(we omit to write down the other principal typing inference rules)
(¶) The premises of the form $\Vdash \cdots$ share no variables.

We can prove that all the above are admissible. So,

Theorem 4. *The linear λ-calculus admits the principal typing theorem.*

6 Concluding Remarks

Mackie [11] introduced a version of linear λ-calculus, a translation from the calculus to a proof structure, and studied efficient implementation of call-by-(name/value/need) evaluation of the λ-calculus.

By using LCHAM and the extension, we will analyze computation of linear λ-calculi neatly. Then we will study the (sub)computation can be encapsulated (and parallelized) in recent concurrent calculi.

References

1. S. Abramsky. Computational interpretations of linear logic. *TCS*, 111:3–57, 1993.
2. G. Bellin and P. Scott. On the π-calculus and linear logic. *TCS*, 135:11–65, 1994.
3. N. Benton, G. Bierman, J. Martin E. Hyland, and V. de Paiva. A term calculus for intuitionistic linear logic. In M. Bezem and J. F. Groote, eds., *Typed Lambda Calculi and Applications, Proceedings*, vol. 664 of *LNCS*, pp. 75–90. 1993.
4. N. Benton and P. Wadler. Linear logic, monads and the lambda calculus. In *Proceedings of the 11th LICS*, pp. 420–431, 1996.
5. G. Berry and G. Boudol. The chemical abstract machine. In *Conference Record of the 17th POPL*, pp. 81–94 1990.
6. L. Cardelli and A. D. Gordon. Mobile Ambients. In M. Nivat, ed., *Foundations of Software Science and Computational Structures*, vol. 1378 of *LNCS*, pp. 140–155, 1998.
7. J. Chirimar, C. A. Gunter, and J. G. Riecke. Proving memory management invariants for a language based on linear logic. In *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, pp. 139–150. 1992.
8. C. Fournet and G. Gonthier. The reflexive cham and the join-calculus. In *Conference Record of the 23rd POPL*, pp. 372–385, 1996.
9. J.-Y. Girard. Linear logic. *TCS*, 50:1–102, 1987.
10. M. Hasegawa. Categorical Glueing and Logical Predicates for Models of Linear Logic. Research Institute of Mathematical Sciences, Preprint RIMS-1223, 1999.
11. I. Mackie, The Geometry of Implementation, Imperial College of Science, 1994.
12. I. Mackie. Lilac — a functional programming language based on linear logic. *JFP*, 4(4):395–433, 1994.
13. J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *TCS, special issue on papers presented at MFPS'95.*
14. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.